15-110: Principles of Computing

Lecture 3: Simplifying instructions and abstraction

August 07, 2022

Clear algorithms

- **Why is it important for algorithms to be clear?**
- Writing a sequence of steps in English (or in any natural language) can quickly become cumbersome
- More “formal” and less ambiguous language

➡ Simplifications

Variables

- In math we commonly use **names** to refer to values

Variables

- In math we commonly use **names** to refer to values
- **Variables:** provide a way to name *information and* access and modify the information by using the name
- A named *container* of information

➢ What can we do with a variable (e.g., x)?

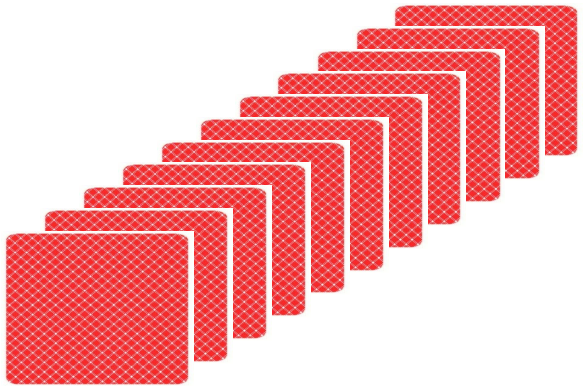
- ✓ **Assign** its value
- ✓ **Read / use** its value
- ✓ **Modify** its value

$$x := 2$$
$$y := x + 2$$
$$x := 4.5$$

Indices

- Sometimes we would like to give names to sequences of objects

Card Deck



$$c_0, c_1, c_2, c_3, \ldots, c_{n-1}$$

c_i Card at the ith position in the pile

Finding the maximum card

1. Pick up the first card from the *deck* pile (n cards)
2. Record down the number v and remove the card from the deck (put it in *done* pile)
3. Assign the number v to max value

4. Pick up the next card from the deck
5. Look at the number, v, and remove the card from the deck
6. If the number is higher that current max value: max value becomes v

7. Repeat 4-6 $n-1$ times (i.e., until no more cards in deck)
8. Output max value
9. Stop

Variables and indices at work

1. Pick up the first card from the *deck* pile (n cards)
2. Record down the number v and remove the card from the deck (put it in *done* pile)
3. Assign the number v to max value

4. Pick up the next card from the deck
5. Look at the number, v, and remove the card from the deck
6. If the number is higher that current max value: max value becomes v

7. Repeat 4-6 $n-1$ times (i.e., until no more cards in deck)
8. Output max value
9. Stop

1. Define a **variable** to hold the number of cards: n, e.g., $n = 52$
2. Label the cards values with a set of **indices**: $c_0, c_1, c_2, \cdots, c_{n-1}$
3. Define a **variable** max to hold the best value so far
4. $\max := c_0$
5. Card **index variable**, initialized to 1: $i := 1$
6. Check if $i < n$:
7. if yes: Check if $c_i > max$
8. if yes: $\max := c_i;\ i := i + 1$; go back to step 6
9. if no: $i := i + 1$; go back to step 6
10. If no: highest card is max

Using a *Repeat for* directive

1. Input: Let n be the number of cards
2. Input: Let $c_0, c_1, c_2, \cdots, c_{n-1}$ be the card values
3. Let max be the highest card we have seen
4. $\max := c_0$
5. Let i be an **index variable**
6. **Repeat for** i := 1, … , n-1
7. if yes: Check if $c_i > max$
8. if yes: $\max := c_i$
9. **Output:** highest card is max

Before:

1. Define a **variable** to hold the number of cards: n, e.g., $n = 52$
2. Label the cards values with a set of **indices**: $c_0, c_1, c_2, \cdots, c_{n-1}$
3. Define a **variable** max to hold the best value so far
4. $\max := c_0$
5. Card **index variable**, initialized to 1: $i := 1$
6. Check if $i < n$:
7. if yes: Check if $c_i > max$
8. if yes: $\max := c_i$; $i := i + 1$; go back to step 6
9. if no: $i := i + 1$; go back to step 6
10. If no: highest card is max

What the **Repeat for** directive does?

Abstraction

Story 1 Example:

First you take your ___bread___ then add a layer of ___butter___
before you pour on a hearty dose of ___jelly___ .
Next, press some ___chips___ down into the ___bread___ before
covering with a sprinkle of ___pepper___ .
That's how we make a ___sandwich___ !

Find another problem that can be solved using the same algorithm. Be creative!

First you take your _____ then add a layer of _____
before you pour on a hearty dose of _____ .
Next, press some _____ down into the _____ before
covering with a sprinkle of _____ .
That's how I make a _____ !